Applications of Category Theory in Computer Science and Physics

Tawhid Bin Omar

"The purpose of category theory is to show that which is trivial is trivially trivial."
— Peter Freyd

March 7, 2026

1 Introduction

Category theory is often called "the mathematics of mathematics." Imagine high school algebra, where you study numbers and how they relate. Now, zoom out. Instead of numbers, imagine entire mathematical systems—like all possible groups, or all topological spaces—as single points.

Category theory provides the language to describe how these massive structures relate to one another. It shifts the focus from *what things are* (objects) to *how they interact* (morphisms). If set theory is about the dots, category theory is about the arrows between them. This shift in perspective has revolutionized computer science (giving us robust type systems) and physics (organizing quantum mechanics), offering a unified high-level language for structure itself.

2 Foundations: Basic Definitions

2.1 Categories

Think of a category as a simplified map of a network. You have cities (objects) and roads (morphisms) connecting them.

- You can always stay in your current city (identity).
- If you can go from City A to City B, and B to C, you can combine those trips to go from A to C (composition).
- The path A $\to$ B $\to$ C is the same valid route regardless of whether you plan A $\to$ B first or B $\to$ C first (associativity).

Formally, we strip away the "cities" and "roads" and just talk about objects and arrows:

Definition 2.1: Category

A **category C** consists of:

1. A collection $\mathrm{ob}(\mathbf{C})$ of **objects**
2. For each pair of objects $A, B \in \mathrm{ob}(\mathbf{C})$, a collection $\mathbf{C}(A, B)$ of **morphisms** (or **arrows**) from A to B
3. For each object A, an **identity morphism** $\mathrm{id}_A \in \mathbf{C}(A, A)$
4. A **composition operation**: for morphisms $f \in \mathbf{C}(A, B)$ and $g \in \mathbf{C}(B, C)$, there exists a composite $g \circ f \in \mathbf{C}(A, C)$

subject to the following axioms:

- **Associativity**: For $f : A \to B$, $g : B \to C$, $h : C \to D$, we have $h \circ (g \circ f) = (h \circ g) \circ f$
- **Identity**: For any $f : A \to B$, we have $f \circ \mathrm{id}_A = f = \mathrm{id}_B \circ f$

Example 2.2: Standard Categories

1. **Set**: Objects are sets, morphisms are functions
2. **Grp**: Objects are groups, morphisms are group homomorphisms
3. **Top**: Objects are topological spaces, morphisms are continuous functions
4. $\mathbf{Vect}_k$: Objects are vector spaces over field k, morphisms are linear transformations

2.2 Functors: The Translators

If categories are different mathematical "worlds" (like the world of Sets or the world of Groups), a **Functor** is a bridge between them. It translates objects from one world to another, but crucially, it also translates the *relationships* (arrows) between them.

Imagine a specialized dictionary that doesn't just translate words, but ensures sentences still make grammatical sense in the new language. That is a functor: it preserves the structure of the original category inside the new one.

Definition 2.3: Functor

Let $\mathbf{C}$ and $\mathbf{D}$ be categories. A **functor** $F : \mathbf{C} \to \mathbf{D}$ consists of:

1. An **object function**: for each object $A \in \text{ob}(\mathbf{C})$, an object $F(A) \in \text{ob}(\mathbf{D})$
2. A **morphism function**: for each morphism $f : A \to B$ in $\mathbf{C}$, a morphism $F(f) : F(A) \to F(B)$ in $\mathbf{D}$

satisfying:

- $F(\text{id}_A) = \text{id}_{F(A)}$ for all objects A
- $F(g \circ f) = F(g) \circ F(f)$ for all composable morphisms

Example 2.4: Important Functors

1. **Forgetful Functor** $U : \mathbf{Grp} \to \mathbf{Set}$: "forgets" group structure
2. **Free Functor** $F : \mathbf{Set} \to \mathbf{Grp}$: sends a set to the free group it generates
3. **Fundamental Group** $\pi_1 : \mathbf{Top}_* \to \mathbf{Grp}$: topological space to fundamental group

Comparing Translations

If functors are translations between worlds, **natural transformations** are comparisons between translations.

Suppose you have two different ways to translate English to French (Functor F and Functor G). A natural transformation is a rule for sliding from one translation to the other seamlessly, ensuring that the meaning is preserved no matter which path you take. It relates the image of one functor to the image of another.

The true genius of category theory emerges here: natural transformations are "morphisms between morphisms between categories."

Definition 2.5: Natural Transformation

Let $F, G : \mathbf{C} \to \mathbf{D}$ be functors. A **natural transformation** $\eta : F \Rightarrow G$ is a family of morphisms $\{\eta_A : F(A) \to G(A)\}_{A \in \mathrm{ob}(\mathbf{C})}$ indexed by objects of $\mathbf{C}$, such that for every morphism $f : A \to B$ in $\mathbf{C}$, the following diagram commutes:

$$\begin{array}{ccc} F(A) & \xrightarrow{\eta_A} & G(A) \\ {\scriptstyle F(f)}\downarrow & & \downarrow{\scriptstyle G(f)} \\ F(B) & \xrightarrow[\eta_B]{} & G(B) \end{array}$$

The condition means: $\eta_B \circ F(f) = G(f) \circ \eta_A$ (naturality square).

Theorem 2.6: Functor Category

For categories $\mathbf{C}$ and $\mathbf{D}$, there exists a **functor category** $[\mathbf{C}, \mathbf{D}]$ where:

- Objects are functors $F : \mathbf{C} \to \mathbf{D}$
- Morphisms are natural transformations between functors

This is a profound idea: functors and natural transformations themselves form a category!

3 Key Concepts and Universal Properties

3.1 Universal Properties

The concept of a universal property is central to categorical thinking. Rather than defining an object by its internal structure, we define it by its relationships to all other objects.

Definition 3.1: Initial Object

An object I in a category $\mathbf{C}$ is **initial** if for every object $A \in \mathrm{ob}(\mathbf{C})$, there exists a unique morphism $I \to A$.

Definition 3.2: Terminal Object

An object T in a category $\mathbf{C}$ is **terminal** if for every object $A \in \mathrm{ob}(\mathbf{C})$, there exists a unique morphism $A \to T$.

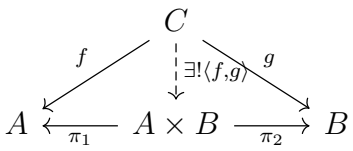
Example 3.3: Universal Objects in Set

- The empty set $\emptyset$ is initial (unique function to any set: the empty function)
- Any singleton set $\{*\}$ is terminal (unique function from any set: map everything to $*$)

3.2 Products and Coproducts

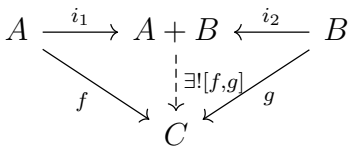
Definition 3.4: Product

A **product** of objects A and B in category $\mathbf{C}$ is an object $A \times B$ together with projection morphisms $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$ such that:
For any object C with morphisms $f : C \to A$ and $g : C \to B$, there exists a unique morphism $\langle f, g \rangle : C \to A \times B$ making the following diagram commute:



Definition 3.5: Coproduct

A **coproduct** of objects A and B is an object $A + B$ (also written $A \sqcup B$) together with injection morphisms $i_1 : A \to A + B$ and $i_2 : B \to A + B$ satisfying the dual universal property (arrows reversed).



Example 3.6: Products and Coproducts

- In **Set**: Product is Cartesian product, coproduct is disjoint union
- In $\mathbf{Vect}_k$: Both are the direct sum $V \oplus W$
- In type theory: Product is tuple `(a,b)`, coproduct is `Either a b`

3.3 Adjunctions: The Heart of Category Theory

Adjunctions are perhaps the most important concept in category theory.

Definition 3.7: Adjunction

Let $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$ be functors. We say F is **left adjoint** to G (written $F \dashv G$) if there exists a natural bijection:

$$\mathbf{D}(F(A), B) \cong \mathbf{C}(A, G(B))$$

for all objects $A \in \mathbf{C}$ and $B \in \mathbf{D}$.

Equivalently, there exist natural transformations:

- $\eta : \mathrm{id}_{\mathbf{C}} \Rightarrow G \circ F$ (unit)
- $\varepsilon : F \circ G \Rightarrow \mathrm{id}_{\mathbf{D}}$ (counit)

satisfying the triangle identities.

Example 3.8: Fundamental Adjunctions

1. **Free-Forgetful**: $F : \mathbf{Set} \to \mathbf{Grp}$ (free group) $\dashv U : \mathbf{Grp} \to \mathbf{Set}$ (forgetful). A function from S to underlying set of G extends uniquely to a homomorphism $F(S) \to G$.
2. **Currying**: $(- \times A) \dashv (-)^A$ gives $\mathrm{Hom}(C \times A, B) \cong \mathrm{Hom}(C, B^A)$

Lemma 3.9: Adjoints Preserve Limits/Colimits

1. Left adjoints preserve colimits
2. Right adjoints preserve limits

Programmable Semicolons

In computer science, a **Monad** is often described as a "design pattern" or a "wrapper." It represents a specific computational context—like a computation that might fail (Maybe), or one that has side effects (IO).

Intuitively, a Monad allows you to chain operations together. If you have a value wrapped in a context (like 'Maybe Int'), you can't just apply a normal function to it. You need a way to "unwrap" it, apply the function, and "rewrap" it. Monads provide the standard interface for doing this safely

3.4 Monads: Abstraction of Computation

Monads unify many computational patterns and are crucial in both mathematics and programming.

Definition 3.10: Monad

A **monad** on a category $\mathbf{C}$ is a triple (T, η, μ) where:

- $T : \mathbf{C} \to \mathbf{C}$ is an endofunctor
- $\eta : \mathrm{id}_{\mathbf{C}} \Rightarrow T$ is a natural transformation (unit)
- $\mu : T \circ T \Rightarrow T$ is a natural transformation (multiplication)

satisfying:

- **Associativity**: $\mu \circ T\mu = \mu \circ \mu T$
- **Unit laws**: $\mu \circ T\eta = \mu \circ \eta T = \mathrm{id}_T$

$$\begin{array}{ccc} TTT & \xrightarrow{T\mu} & TT \\ {\scriptstyle \mu T}\downarrow & & \downarrow{\scriptstyle \mu} \\ TT & \xrightarrow[\mu]{} & T \end{array} \qquad \begin{array}{ccccc} T & \xrightarrow{\eta T} & TT & \xleftarrow{T\eta} & T \\ & \searrow^{\mathrm{id}} & \downarrow{\scriptstyle \mu} & \swarrow_{\mathrm{id}} & \\ & & T & & \end{array}$$

Theorem 3.11: Monads from Adjunctions

Every adjunction $F \dashv G$ gives rise to a monad $T = G \circ F$ with unit η from the adjunction unit and multiplication $\mu = G\varepsilon F$ from the adjunction counit.

Example 3.12: Common Monads

In Haskell: **Maybe** (failure), **List** (nondeterminism), **State** (stateful computations), **IO** (side effects). In mathematics: powerset monad $\mathcal{P}$, free group monad, distribution monad.

4 Key Results

4.1 The Yoneda Lemma

The Yoneda Lemma is one of the most important results in category theory.

Theorem 4.1: Yoneda Lemma

Let $\mathbf{C}$ be a locally small category, $A \in \mathrm{ob}(\mathbf{C})$, and $F : \mathbf{C}^{op} \to \mathbf{Set}$ a functor. Then there is a bijection:

$$\mathrm{Nat}(\mathrm{Hom}_{\mathbf{C}}(-, A), F) \cong F(A)$$

natural in both A and F. Moreover, this bijection is given by evaluating a natural transformation at id_A.

Proof Sketch. Define $\Phi : \mathrm{Nat}(\mathrm{Hom}(-, A), F) \to F(A)$ by $\Phi(\alpha) = \alpha_A(\mathrm{id}_A)$. For the inverse, given $x \in F(A)$, define $\alpha^x_B(f) = F(f)(x)$. Naturality and bijectivity follow by direct computation. □ □

Corollary 4.2: Yoneda Embedding

The functor $h : \mathbf{C} \to [\mathbf{C}^{op}, \mathbf{Set}]$ defined by $h(A) = \mathrm{Hom}(-, A)$ is fully faithful. This means $\mathbf{C}$ embeds into its category of presheaves.

4.2 Applications in Computer Science and Physics

Curry-Howard-Lambek Correspondence: Category theory, type theory, and logic are deeply connected. In programming: products are tuples, coproducts are sum types (`Either`), exponentials are functions. The adjunction $(- \times A) \dashv (-)^A$ explains currying. Monads in Haskell (Maybe, State, IO) structure computational effects.

Quantum Mechanics: The category **Hilb** of Hilbert spaces provides a framework for quantum theory. Tensor products model composite systems. Quantum circuits live in monoidal categories where objects are qubit counts and morphisms are gates.

TQFT: A topological quantum field theory is a functor $Z : \mathbf{Cob}_n \to \mathbf{Vect}$ from cobordisms to vector spaces, capturing topological invariants through functorial properties.

5 Exercises

Exercise 5.1: Prove that if I is an initial object in category $\mathbf{C}$, then any two initial objects are uniquely isomorphic.

Exercise 5.2: Verify the monad laws for the List monad in Haskell where `return x = [x]` and `xs >>= f = concat (map f xs)`.

6 Conclusion

Category theory provides a unifying framework through **abstraction**, **composition**, **universality**, and **duality**. Focus on morphisms over objects, seek universal properties over explicit constructions, think compositionally, and look for adjunctions.
Further Reading: Mac Lane's *Categories for the Working Mathematician*, Milewski's *Category Theory for Programmers*, nLab (ncatlab.org).